

# Post-quantum Cryptography at Google

**Stefan Kölbl**

Symposium PQC, Netherlands

June 13th, 2023

# Agenda

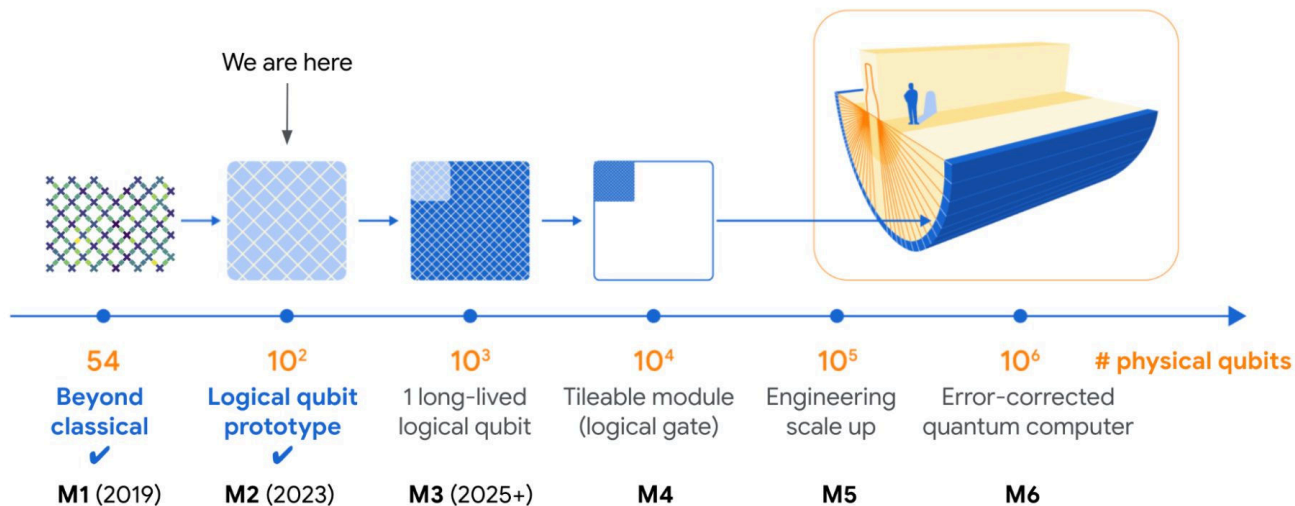
1. Our view on post-quantum cryptography
2. Deploying PQC at scale
3. Engineering to ease migration





# Part I - Where are we today?

# When will we get a quantum computer?



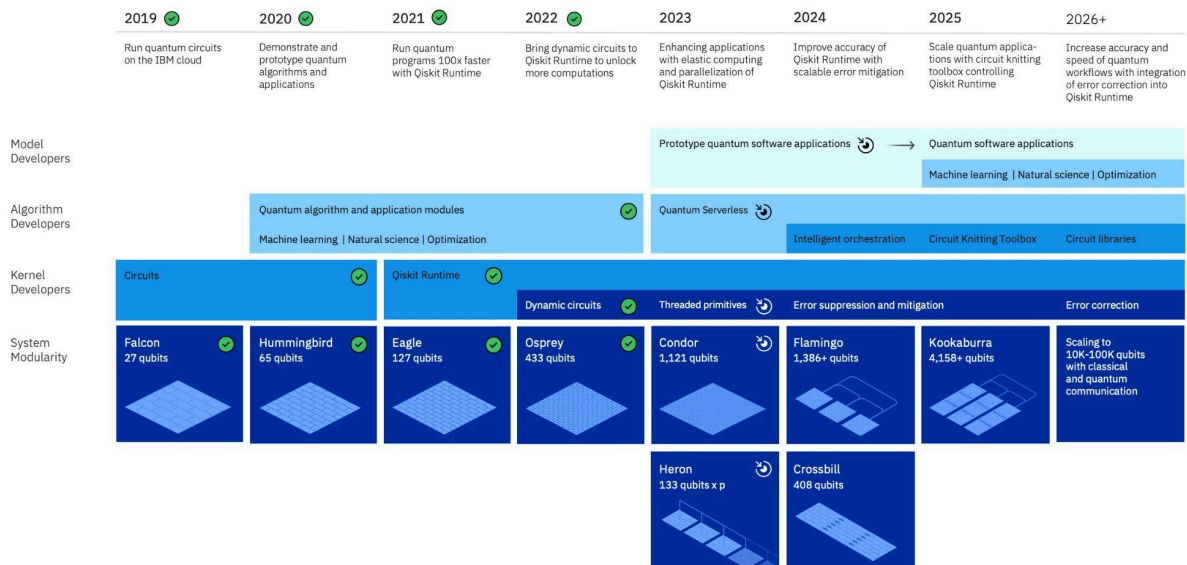
<https://ai.googleblog.com/2023/02/suppressing-quantum-errors-by-scaling.html>

# When will we get a quantum computer?

## Development Roadmap

Executed by IBM   
On target 

IBM Quantum



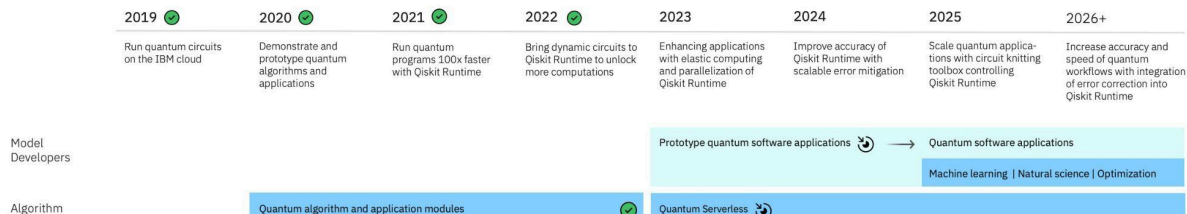
<https://www.ibm.com/quantum/roadmap>

# When will we get a quantum computer?

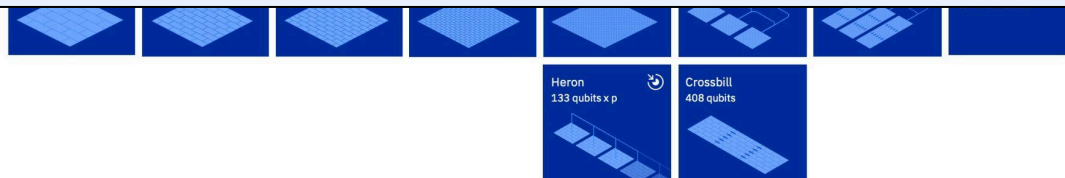
## Development Roadmap

Executed by IBM  
On target

IBM Quantum



To break RSA we will need thousands of \*logical\* qubits



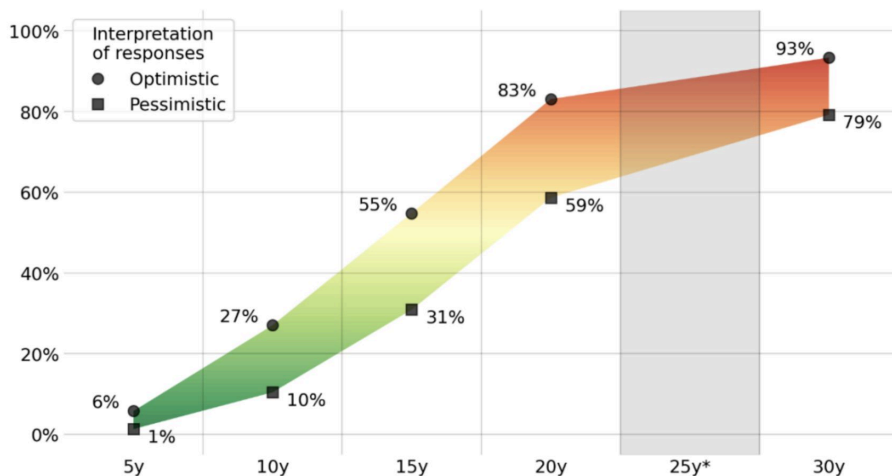
<https://www.ibm.com/quantum/roadmap>

# When will we get a quantum computer?



## 2022 OPINION-BASED ESTIMATES OF THE CUMULATIVE PROBABILITY OF A DIGITAL QUANTUM COMPUTER ABLE TO BREAK RSA-2048 IN 24 HOURS, AS FUNCTION OF TIMEFRAME

Estimates of the cumulative probability of a cryptographically-relevant quantum computer in time: range between average of an optimistic (top value) or pessimistic (bottom value) interpretation of the estimates indicated by the respondents. [\*Shaded grey area corresponds to the 25-year period, not considered in the questionnaire.]



<https://globalriskinstitute.org/publication/2022-quantum-threat-timeline-report/>

A close-up photograph of a quantum processor chip, likely a Sycamore processor. The chip is dark blue with a complex network of fine, parallel lines representing qubits and their connections. The text "Google AI Quantum" and "Sycamore" is visible on the chip. The background is a warm, golden-brown color with some abstract patterns. A blue arrow points towards the bottom right corner of the image.

Why care about a  
threat that does not  
exist yet?



# Why we care about this today



Impact of  
cryptography failing



Migration takes a  
long time



Store now – decrypt  
later

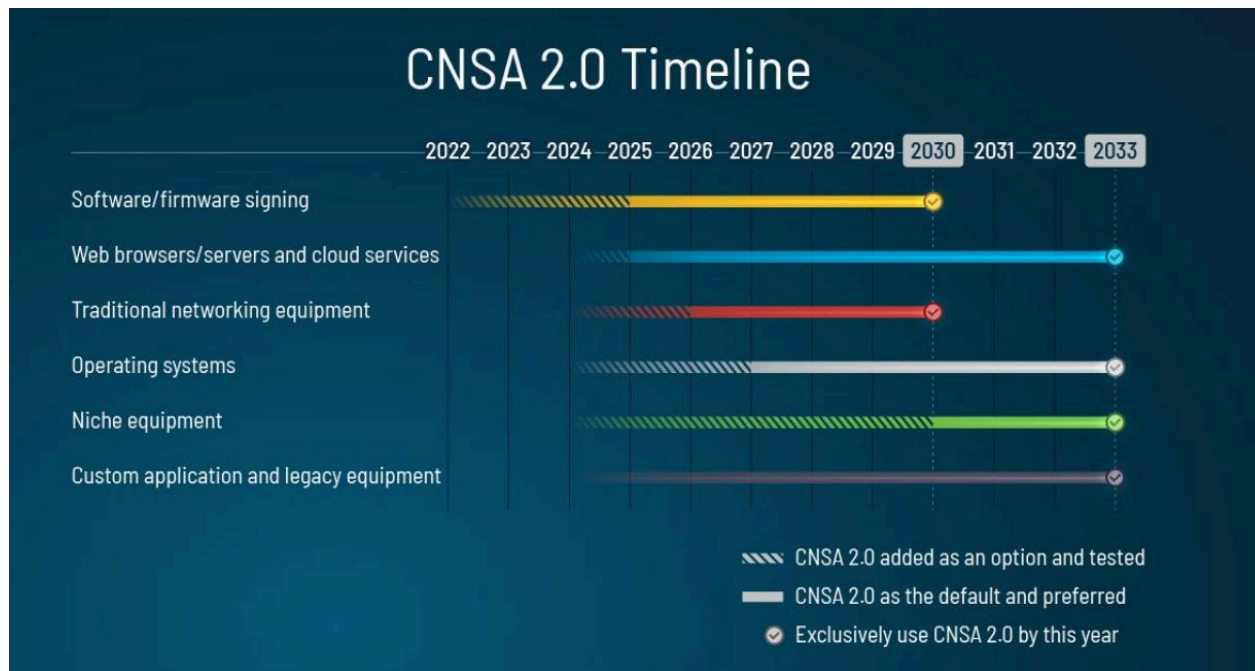
# PQC Standards

## Algorithms

- **IETF RFC-8391, RFC-8554, ISO 14888-4 & NIST SP 800-208**
  - Scope: Stateful Hash-Based Signatures
  - Status: Published (ISO in DIS stage)
- **NIST PQC Competition**
  - Scope: KEMs, Signatures
  - Status: Drafts soon, standards ~Mid 2024
- **ISO/IEC**
  - Scope: KEMs
  - Status: Recently started



# PQC Standards



# PQC Challenges - Hardware

## Devices

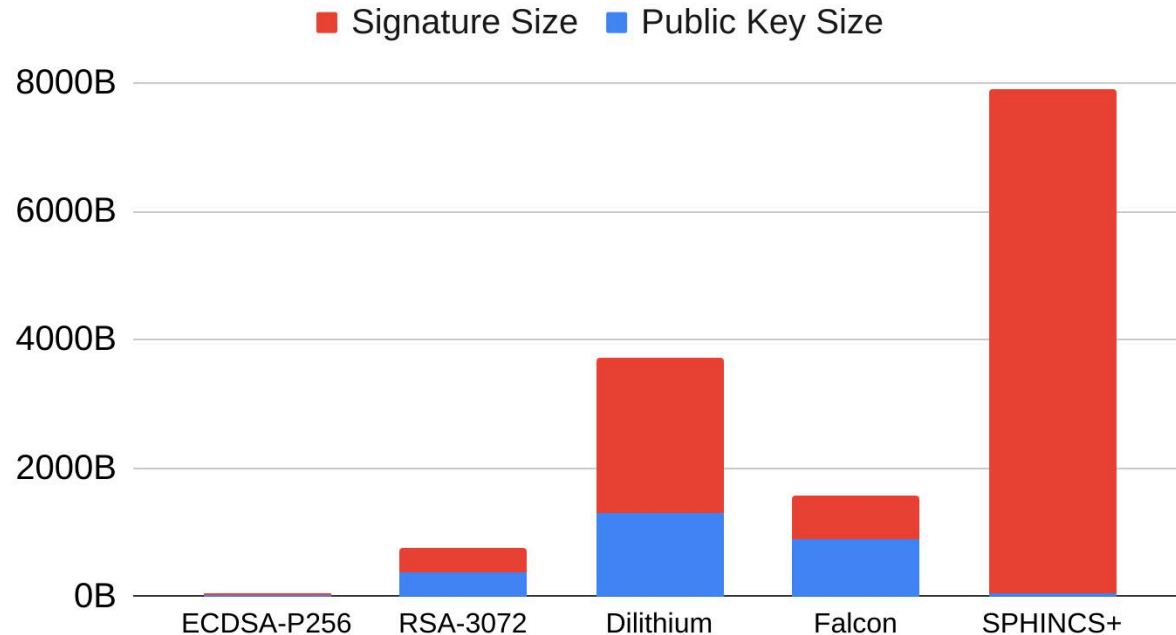
- Root of trust
- Security keys
- Consumer devices

## Challenges

- Long development cycles
- Long devices lifetime
- Difficulty to upgrade
- Resource constrained environments


# PQC Challenges - Performance

Public Key Size, Signature Size



# PQC Challenges - Performance

Challenges with *fancy* crypto:

- Blind Signatures
- (V)OPRF (e.g. Privacy Pass  )
  - Communication overhead >100MB\*

**Huge performance gap between classic <-> PQC**

\*See <https://eprint.iacr.org/2023/232>

# PQC Challenges - State management

LMS/XMSS (stateful hash-based signatures)

- Only makes sense if SPHINCS+ performance is prohibitive.
- State management complicates signing infrastructure significantly.

Slowly sees adoption:

- Recommended by [CNSA 2.0](#), [ANSI](#) (France), [BSI](#) (Germany)
- Already used in: [OpenSSH](#), [mbedTLS](#), [Infineon TPM](#)

# PQC Priorities

As a large organization, where do you even start?



# PQC Priorities

- 1) Encryption in Transit
- 2) Signatures, when Public Keys are hard to change
- 3) All other Asymmetric Cryptography
- 4) Symmetric Cryptography



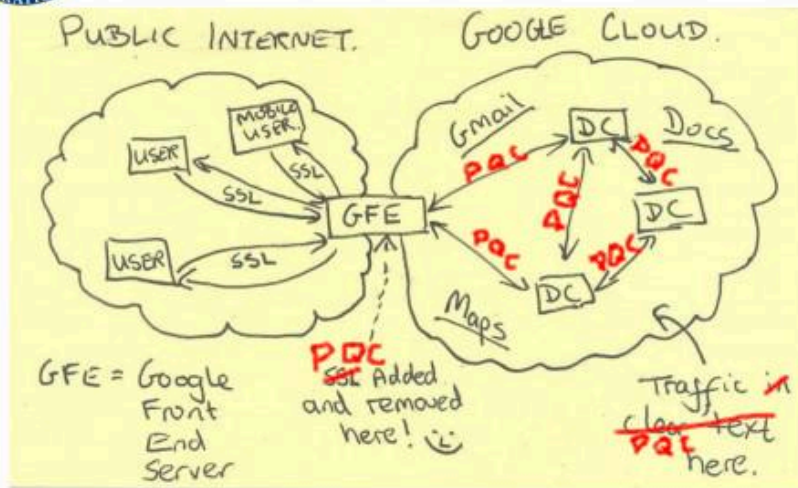
# Part II - Encryption in Transit

# ALTS: Overview

TOP SECRET//SI//NOFORN



## Current Efforts - Google



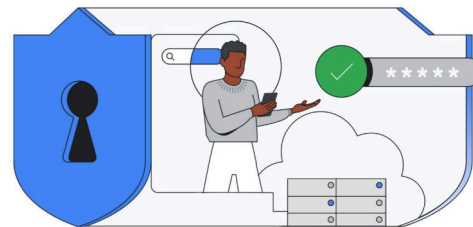
TOP SECRET//SI//NOFORN

# Bringing PQC to ALTS

- ALTS protects all our internal traffic
- Ideal candidate for early PQC adoption
- Enabled a PQC algorithm
  - Deployed in Hybrid mode
  - X25519 + HRSS

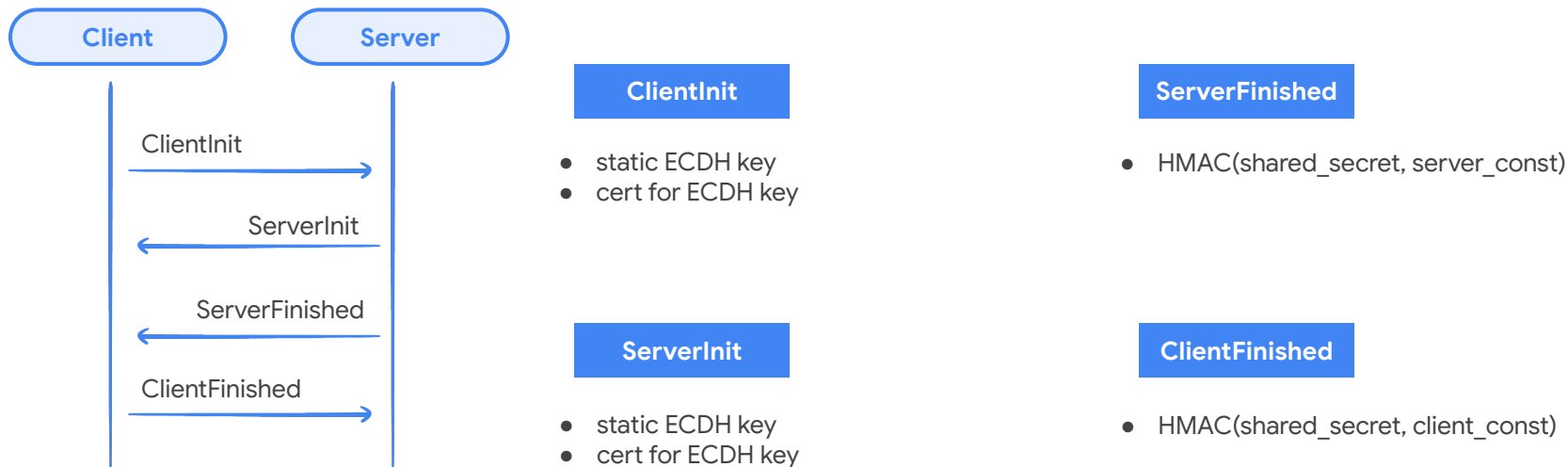
Security & Identity  
**Securing tomorrow today: Why Google now protects its internal communications from quantum threats**

November 19, 2022



<https://cloud.google.com/blog/products/identity-security/why-google-now-uses-post-quantum-cryptography-for-internal-comms/>

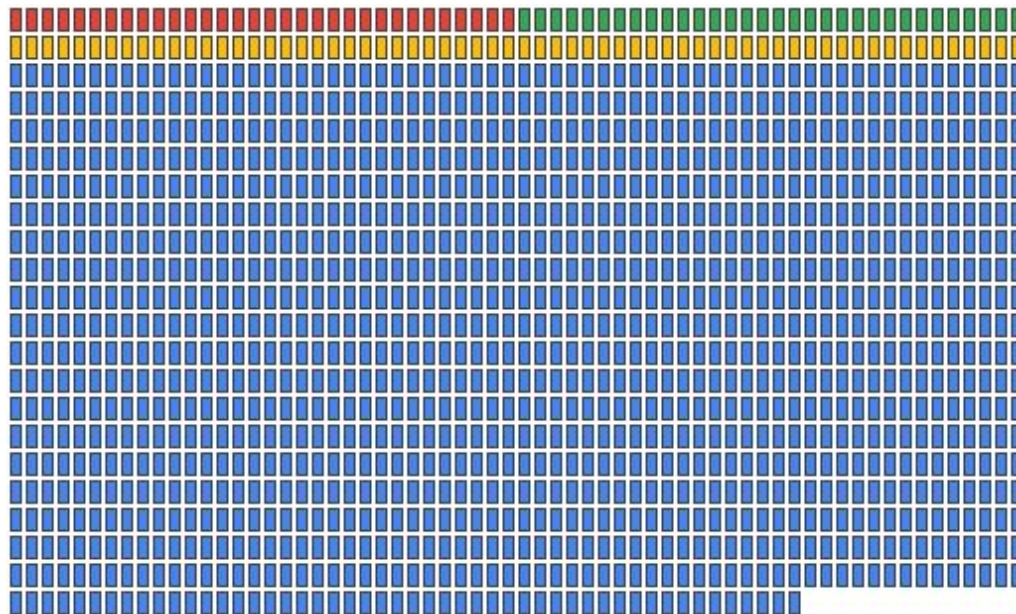
# ALTS: Overview



# ALTS: Overview

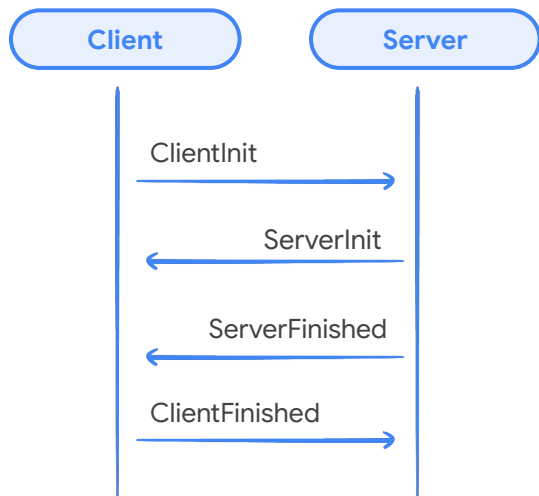


# PQC Overview



- Protocol Overhead (estimate)
- X25519 Keyshare
- Certificate
- HRSS public key/ciphertext

# ALTS PQC



## ClientInit

- static ECDH key
- cert for ECDH key
- ephemeral PQC public key

## ServerInit

- static ECDH key
- cert for ECDH key
- PQC KEM ciphertext

## ServerFinished

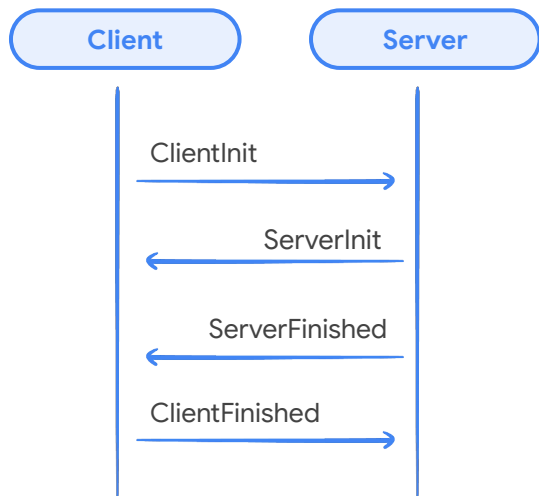
- HMAC(shared\_secret, server\_const)

## ClientFinished

- HMAC(shared\_secret, client\_const)



# ALTS PQC



## ClientInit

- static ECDH key
- cert for ECDH key
- somewhat ephemeral PQC public key

## ServerInit

- static ECDH key
- cert for ECDH key
- PQC KEM ciphertext

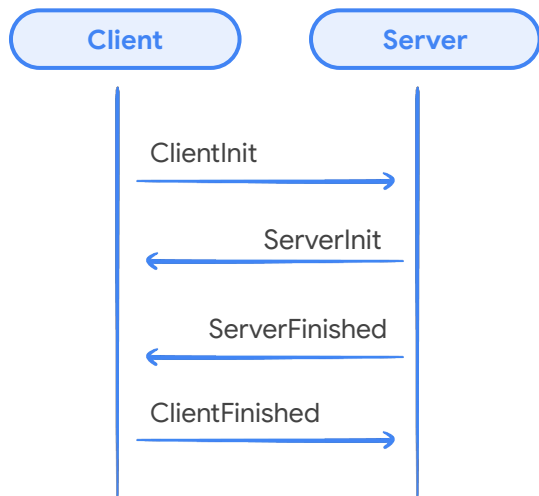
## ServerFinished

- HMAC(shared\_secret, server\_const)

## ClientFinished

- HMAC(shared\_secret, client\_const)

# ALTS PQC



## ClientInit

- static ECDH key
- cert for ECDH key
- resumption ticket
- somewhat ephemeral PQC public key

## ServerInit

- resumption confirmation
- PQC KEM ciphertext

## ServerFinished

- $\text{HMAC}(\text{shared\_secret}, \text{server\_const})$

## ClientFinished

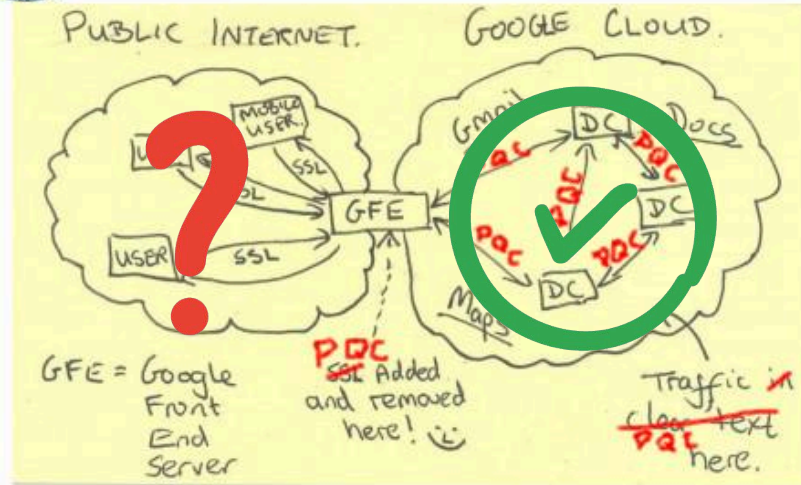
- $\text{HMAC}(\text{shared\_secret}, \text{client\_const})$

# Encryption in transit

TOP SECRET//SI//NOFORN



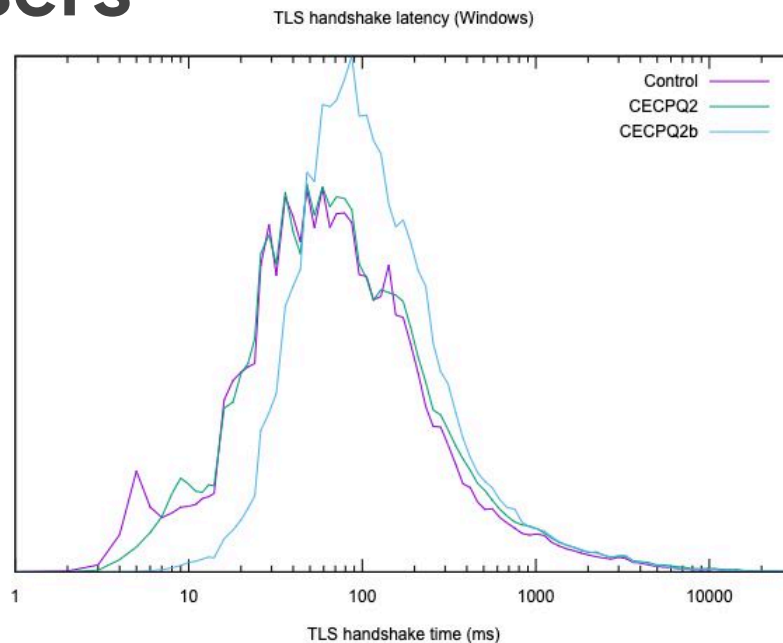
## Current Efforts - Google



TOP SECRET//SI//NOFORN

# Bringing PQC to end-users

- Traffic to user protected by TLS
- Chrome supporting hybrid key-exchange:
  - CECPQ1 (2016)
  - CECPQ2 (2019)

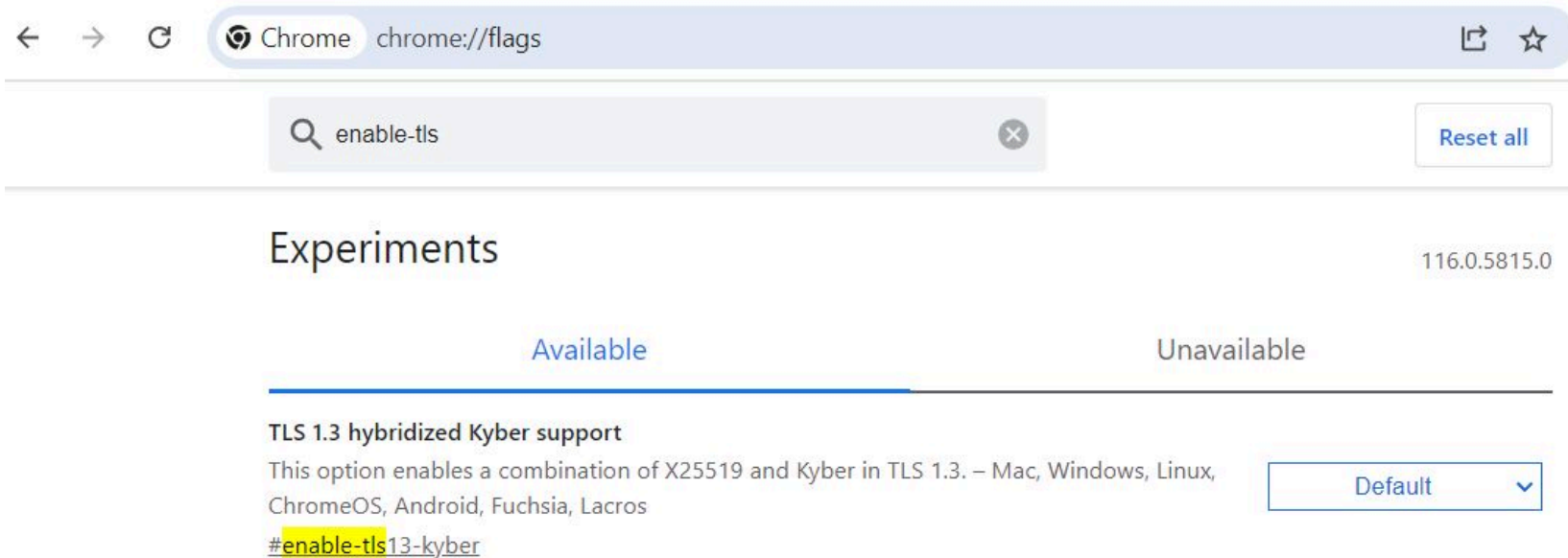


More info at: <https://www.imperialviolet.org/2019/10/30/pqsvssl.html> and <https://blog.cloudflare.com/the-tls-post-quantum-experiment/>

# Bringing PQC to end-users

- Download Chrome Canary
  - <https://www.google.com/chrome/canary/>
- Example test site:
  - <https://pq.cloudflareresearch.com/>

# Bringing PQC to end-users



The screenshot shows the Chrome flags page at chrome://flags. A search bar contains 'enable-tls'. The results are categorized into 'Available' and 'Unavailable'. The 'Available' section shows 'TLS 1.3 hybridized Kyber support' with a description: 'This option enables a combination of X25519 and Kyber in TLS 1.3. – Mac, Windows, Linux, ChromeOS, Android, Fuchsia, Lacros'. The flag is currently set to 'Default'.

chrome://flags

enable-tls

Reset all

Experiments 116.0.5815.0

Available Unavailable

**TLS 1.3 hybridized Kyber support**  
This option enables a combination of X25519 and Kyber in TLS 1.3. – Mac, Windows, Linux, ChromeOS, Android, Fuchsia, Lacros  
[#enable-tls13-kyber](#)

Default

# Bringing PQC to end-users

← → ↻ 🌐 pq.cloudflareresearch.com



## Cloudflare Research: Post-Quantum Key Agreement

On essentially all domains served through [Cloudflare](#), including this one, we have enabled hybrid post-quantum key agreement. Read [our blog](#) for the details.

You are using *X25519Kyber768Draft00* which is **post-quantum secure**.



# Part III - Engineer for agility



# Crypto Agility at Google

Our main goal is to make crypto usable for engineers:

- [RWC'18](#): Achieving high availability in the internal Google KMS.
- [RWC'19](#): Tink: A cryptographic library.
- [RWC'21](#): What's in a key?
- [RWC'23](#): Crypto Agility and Post-Quantum Cryptography

# Tink

- A multi language and multi-platform open source cryptography library.
- <https://github.com/google/tink>



# Tink

Enforce best practices



Don't burden the user



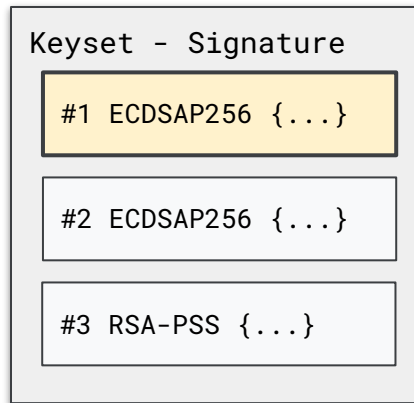
Reliability



# Tink - Keysets

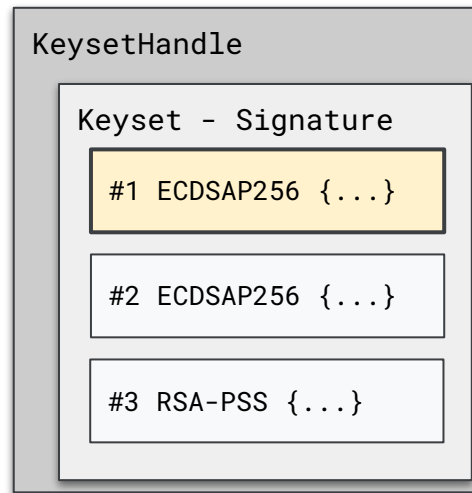
Core concept is that users always use a **keyset**

- A set of keys which implement **the same primitive**.
- Facilitates key rotation.



# Tink - Keyset handle

- Restricts access to sensitive data
- Provides API to obtain a primitive wrapping the keyset, e.g. for signatures:
  - `sign(...)`, uses primary key #1
  - `verify(...)`, finds key to verify



# Example: Signing data

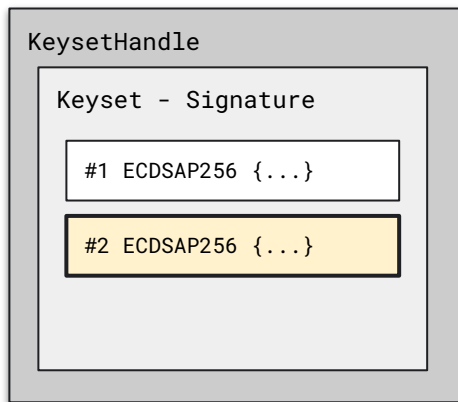
```
import tink
from tink import signature

# Create a keyset with a single key and get a handle to it.
keyset_handle = // Fetch the key from a KMS

# Wrap the keyset into a signing primitive.
sign_primitive = keyset_handle.primitive(signature.PublicKeySign)

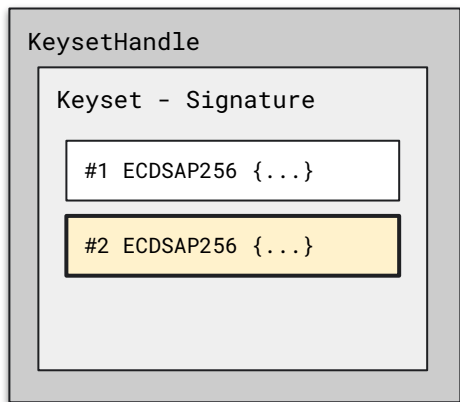
# Use the primitive to sign (uses the primary key!).
signature = sign_primitive.sign("mymessage")
```

# Key Rotation



Key #2 is primary key

# Key Rotation



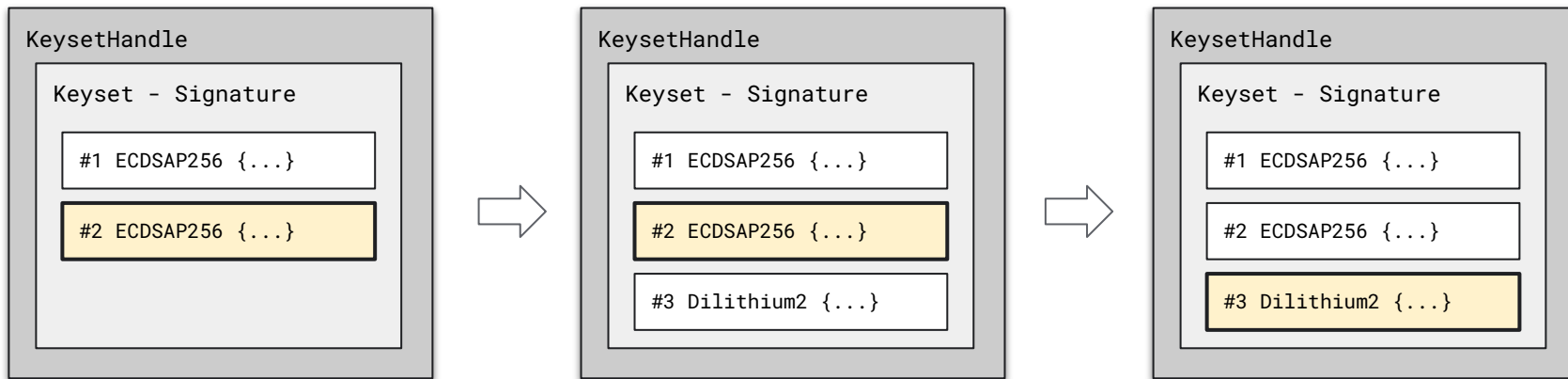
Key #2 is primary key

Tink manages which key is used:

- #2 will sign any new data.
- {#1, #2} will verify signatures.



# Key Rotation



Key #2 is primary key

Key #2 is primary key  
Key #3 is added

Key #3 is primary key

# Key rotation

Key rotation should happen **automatically**

- Forward secrecy.
- Enables speedy recovery from compromise at low operational risk.
- Simplifies switching keys  $\Rightarrow$  Transition to post-quantum crypto.

In practice **hard** to enforce:

- Reliability risks (see [RWC'23](#))

# Takeaways

01

## Rolling out PQC

We started working on ALTS PQC in 2020, and are now finally getting ready to have rolled out PQC for all jobs, with many unforeseen obstacles along the way.

02

## Hybrid deployment

Hybrid deployment allows us to experiment with PQC without risking security regressions. In the worst case, we learned something in an experiment, in the best case we have already mitigated store-now-decrypt later

03

## Invest in tooling

Building the right tools for your engineers can ease complex issues around key management, and make migration easier.



Thank you

